SIAM Conference on Computational Science and Engineering

- MS48 -

Numerical Software for Solving Problems in Computational Science and Engineering

Auxiliary Tools for Data Distribution in Large Numerical Calculations

Osni Marques

Lawrence Berkeley National Laboratory (LBNL)

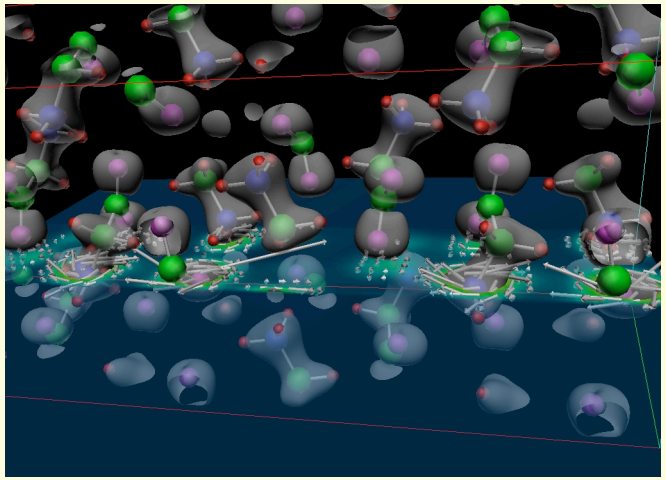
oamarques@lbl.gov

Outline

- ScaLAPACK as a favorite case
- Applications
- Software structure
  - Basic Linear Algebra Subprograms (BLAS)
  - Linear Algebra PACKage (LAPACK)
  - Basic Linear Algebra Communication Subprograms (BLACS)
  - Parallel BLAS (PBLAS)
- Data layout
  - Data distribution
  - Array descriptors
- How to define what is needed?

MS18:
- pMatlab
- Star-P
- –

ScaLAPACK: *Applications*

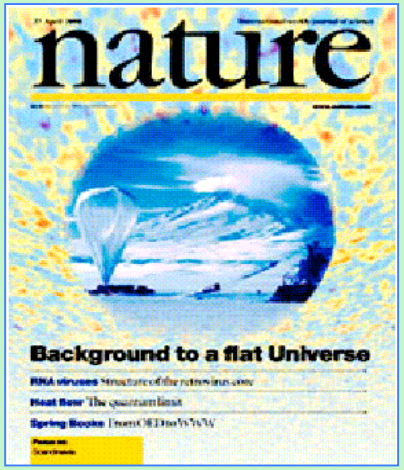


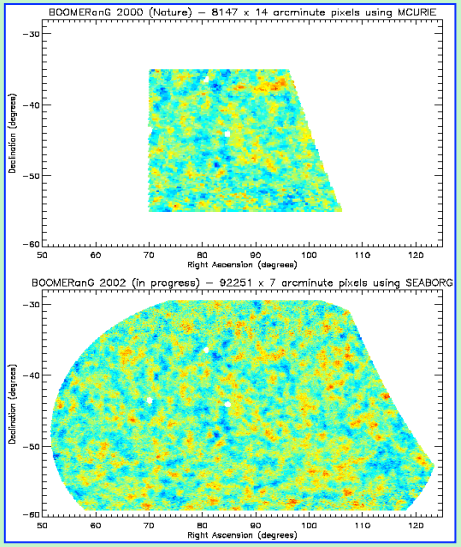
Induced current (white arrows) and charge density (colored plane and gray surface) in crystallized glycine due to an external field (Louie, Yoon, Pfrommer and Canning), eigenvalue problems solved with ScaLAPACK.

Advanced Computational Research in Fusion (SciDAC Project, PI Mitch Pindzola). Point of contact: Dario Mitnik (Dept. of Physics, Rollins College). Mitnik attended the workshop on the ACTS Collection in September 2000. Since then he has been actively using some of the ACTS tools, in particular ScaLAPACK, for which he has provided insightful feedback. Dario is currently working on the development, testing and support of new scientific simulation codes related to the study of atomic dynamics using time-dependent close coupling lattice and time-independent methods. He reports that this work could not be carried out in sequential machines and that ScaLAPACK is fundamental for the parallelization of these codes.

Performance of four science-of-scale applications that use ScaLAPACK functionalities on an IBM SP

Project	Number of processors	Performance (% of peak)
Electromagnetic Wave-Plasma Interactions	1,936	68%
Cosmic Microwave Background Data Analysis	4,096	50%
Terascale Simulations of Supernovae	2,048	43%
Quantum Chromodynamics at High Temperatures	1,024	13%

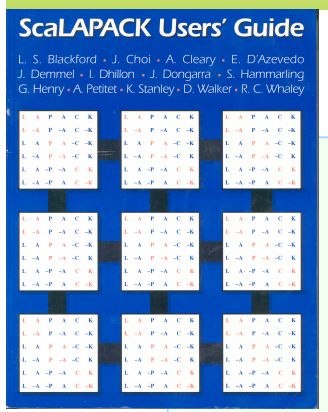




The international BOOMERanG collaboration announced results of the most detailed measurement of the cosmic microwave background radiation (CMB), which strongly indicated that the universe is flat (Apr. 27, 2000). Likelihood methods implemented in the MADCAP software package, using routines from ScaLAPACK, were used to examine the large dataset generated by BOOMERanG.

Office of Science U.S. Department of Energy | Berkeley Lab | ACTS Collection

ScaLAPACK: *software structure*



http://acts.nersc.gov/scalapack

Version 1.7 released in August 2001; recent NSF funding for further development.

ScaLAPACK

PBLAS

Parallel BLAS.

Global

Local

LAPACK

BLACS

Linear systems, least squares, singular value decomposition, eigenvalues.

Communication routines targeting linear algebra operations.

platform specific

BLAS

MPI/PVM/...

Clarity,modularity, performance and portability. Atlas can be used here for automatic tuning.

Communication layer (message passing).

ScaLAPACK: *goals*

- Efficiency
  - Optimized computation and communication engines
  - Block-partitioned algorithms (Level 3 BLAS) for good node performance
- Reliability
  - Whenever possible, use LAPACK algorithms and error bounds.
- Scalability
  - As the problem size and number of processors grow
  - Replace LAPACK algorithm that did not scale (new ones into LAPACK)
- Portability
  - Isolate machine dependencies to BLAS and the BLACS
- Flexibility
  - Modularity: build rich set of linear algebra tools (BLAS, BLACS, PBLAS)
- Ease-of-Use
  - Calling interface similar to LAPACK

BLACS

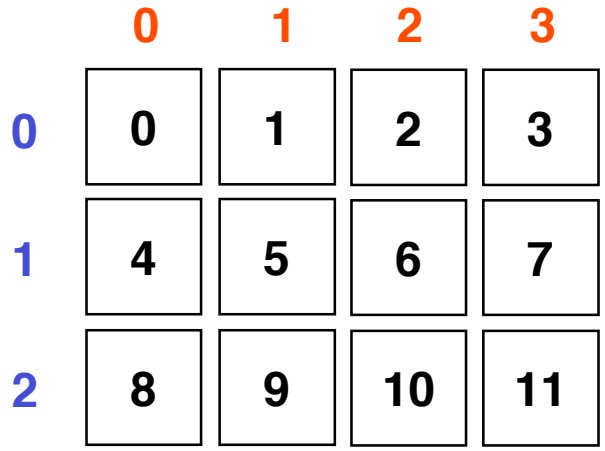
(*Basic Linear Algebra Communication Subroutines*)

- A design tool, they are a conceptual aid in design and coding.
- Associate widely recognized mnemonic names with communication operations. This improves:
  - program readability
  - self-documenting quality of the code.
- Promote efficiency by identifying frequently occurring operations of linear algebra which can be optimized on various computers.

BLACS: *basics*

- Processes are embedded in a two-dimensional grid.

Example: a 3x4 grid



- An operation which involves more than one sender and one receiver is called a *scoped operation*.

Scope	Meaning
Row	All processes in a process row participate.
Column	All processes in a process column participate.
All	All processes in the process grid participate.

BLACS: *example*

```
*   Get system information
    CALL BLACS_PINFO( IAM, NPROCS )
*   Get default system context
    CALL BLACS_GET( 0, 0, ICTXT )
    M
*   Define 1 x (NPROCS/2+1) process grid
    NPROW = 1
    NPCOL = NPROCS / 2 + 1
    CALL BLACS_GRIDINIT( ICTXT, 'Row', NPROW, NPCOL )
    CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
*   If I'm not in the grid, go to end of program
    IF( MYROW.NE.-1 ) THEN
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
        CALL DGESD2D( ICTXT, 5, 1, X, 5, 1, 0 )
      ELSE IF( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
        CALL DGERV2D( ICTXT, 5, 1, Y, 5, 0, 0 )
      END IF
       M
      CALL BLACS_GRIDEXIT( ICTXT )
    END IF
    CALL BLACS_EXIT( 0 )
    END
```

(out) uniquely identifies each process
(out) number of processes available
(in) integer handle indicating the context
(in) use (default) system context
(out) BLACS context
(output) process row and column coordinate
send X to process (1,0)
receive X from process (0,0)
leave context
exit from the BLACS

- **The BLACS context is the BLACS mechanism for partitioning communication space.**
- **A message in a context cannot be sent or received in another context.**
- **The context allows the user to**
  - **create arbitrary groups of processes**
  - **create multiple overlapping and/or disjoint grids**
  - **isolate each process grid so that grids do not interfere with each other**
- **BLACS context ⇔ MPI communicator**

See *http://www.netlib.org/blacs* for more information.

PBLAS

(*Parallel Basic Linear Algebra Subroutines*)

- Similar to the BLAS in portability, functionality and naming:
  - Level 1: vector-vector operations
  - Level 2: matrix-vector operations
  - Level 3: matrix-matrix operations

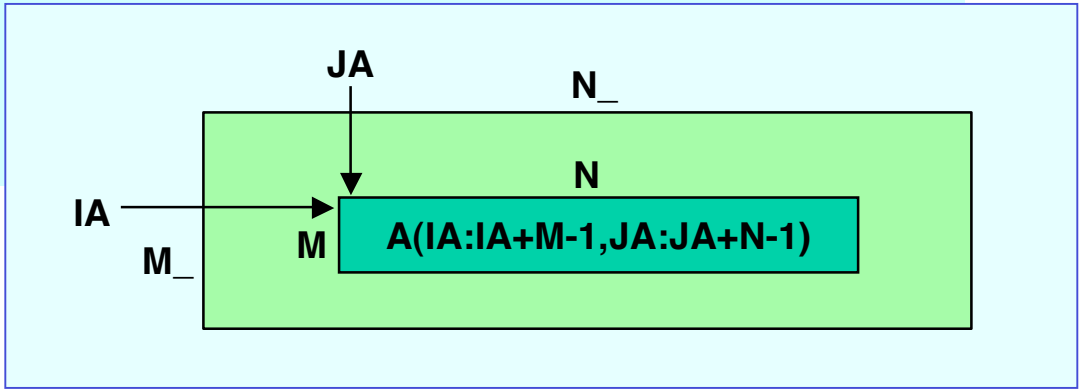
```
CALL DGEXXX( M, N, A( IA, JA ), LDA, ... )
CALL PDGEXXX( M, N, A, IA, JA, DESCA, ... )
```

BLAS

PBLAS

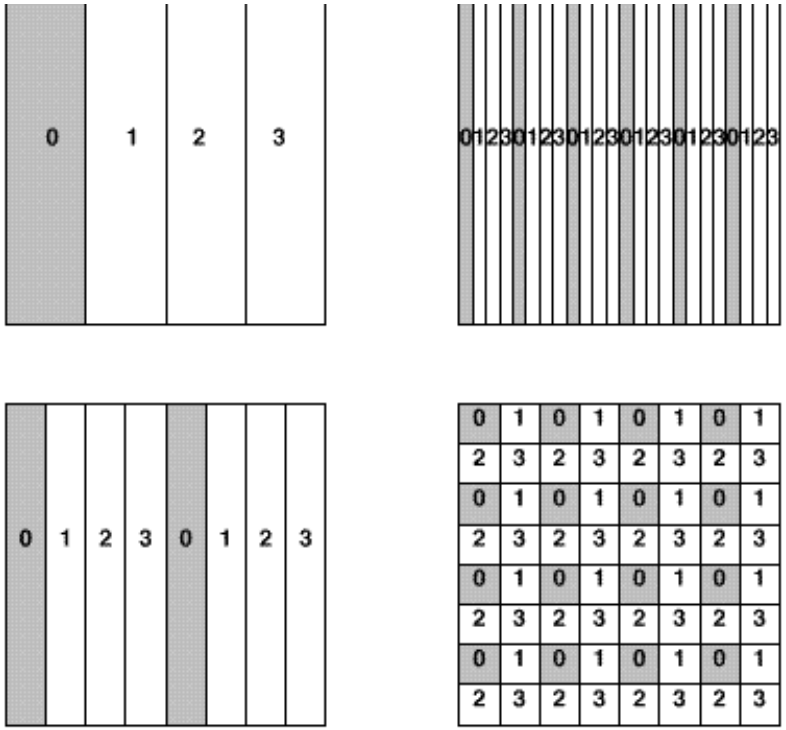
array descriptor (see next slides)

- Built atop the BLAS and BLACS
- Provide global view of the matrix operands



ScaLAPACK: *data layouts*

- 1D block and cyclic column distributions

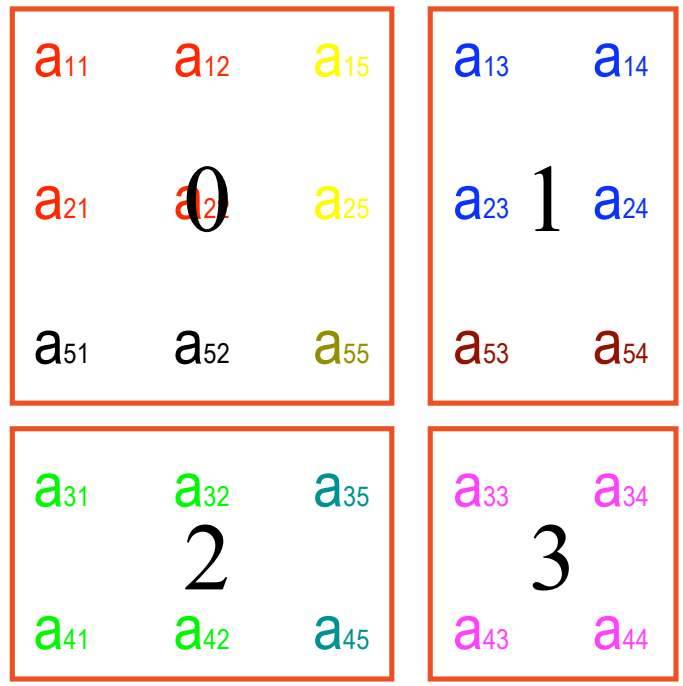


- 1D block-cycle column and 2D block-cyclic distribution
- 2D block-cyclic used in ScaLAPACK for dense matrices

ScaLAPACK: *2D Block-Cyclic Distribution* (1/2)

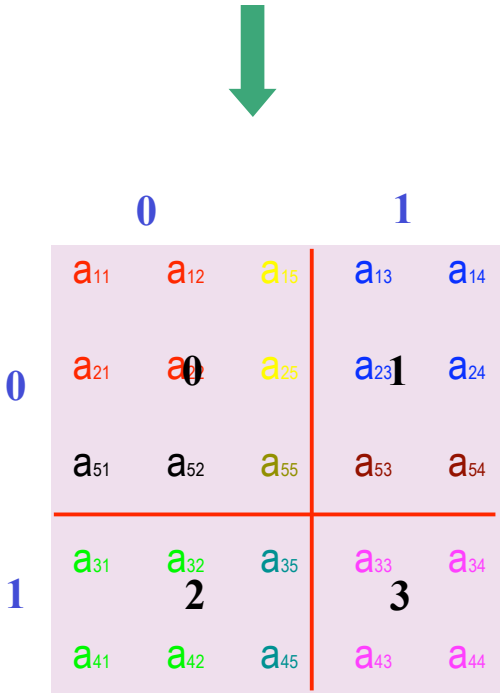
5x5 matrix partitioned in 2x2 blocks

2x2 process grid point of view



ScaLAPACK: *2D Block-Cyclic Distribution (2/2)*

$$\begin{bmatrix} 1.1 & 1.2 & 1.3 & 1.4 & 1.5 \\ -2.1 & 2.2 & 2.3 & 2.4 & 2.5 \\ -3.1 & -3.2 & 3.3 & 3.4 & 3.5 \\ -4.1 & -4.2 & -4.3 & 4.4 & 4.5 \\ -5.1 & -5.2 & -5.3 & -5.4 & 5.5 \end{bmatrix}$$




```
 M

CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )

IF      ( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
        A(1) = 1.1; A(2) = -2.1; A(3) = -5.1;
        A(1+LDA) = 1.2; A(2+LDA) = 2.2; A(3+LDA) = -5.2;
        A(1+2*LDA) = 1.5; A(2+3*LDA) = 2.5; A(3+4*LDA) = -5.5;
ELSE IF ( MYROW.EQ.0 .AND. MYCOL.EQ.1 ) THEN
        A(1) = 1.3; A(2) = 2.3; A(3) = -5.3;
        A(1+LDA) = 1.4; A(2+LDA) = 2.4; A(3+LDA) = -5.4;
ELSE IF ( MYROW.EQ.1 .AND. MYCOL.EQ.0 ) THEN
        A(1) = -3.1; A(2) = -4.1;
        A(1+LDA) = -3.2; A(2+LDA) = -4.2;
        A(1+2*LDA) = 3.5; A(2+3*LDA) = 4.5;
ELSE IF ( MYROW.EQ.1 .AND. MYCOL.EQ.1 ) THEN
        A(1) = 3.3; A(2) = -4.3;
        A(1+LDA) = 3.4; A(2+LDA) = 4.4;
END IF
  M
CALL PDGESVD( JOBU, JOBVT, M, N, A, IA, JA, DESCA, S, U, IU,
              JU, DESCU, VT, IVT, JVT, DESCVT, WORK, LWORK,
              INFO )
  M
```

LDA is the leading dimension of the local array (see next slides)

Array descriptor for A (see next slides)

2D Block-Cyclic Distribution: *pros and cons*

- Ensures good load balance → performance and scalability (analysis of many algorithms to justify this layout).
- Encompasses a large number of data distribution schemes (but not all).
- Needs redistribution routines to go from one distribution to the other.

ScaLAPACK: *array descriptors*

- Each global data object is assigned an *array descriptor*
- The *array descriptor*:
  - Contains information required to establish mapping between a global array entry and its corresponding process and memory location (uses concept of BLACS context).
  - Is differentiated by the DTYPE_ (first entry) in the descriptor.
  - Provides a flexible framework to easily specify additional data distributions or matrix types.
- User must distribute all global arrays prior to the invocation of a ScaLAPACK routine, for example:
  - Each process generates its own submatrix.
  - One processor reads the matrix from a file and send pieces to other processors (may require message-passing for this).

Array Descriptor for Dense Matrices

DESC_()	Symbolic Name	Scope	Definition
1	DTYPE_A	(global)	Descriptor type DTYPE_A=1 for dense matrices.
2	CTXT_A	(global)	BLACS context handle.
3	M_A	(global)	Number of rows in global array A.
4	N_A	(global)	Number of columns in global array A.
5	MB_A	(global)	Blocking factor used to distribute the rows of array A.
6	NB_A	(global)	Blocking factor used to distribute the columns of array A.
7	RSRC_A	(global)	Process row over which the first row of the array A is distributed.
8	CSRC_A	(global)	Process column over which the first column of the array A is distributed.
9	LLD_A	(local)	Leading dimension of the local array.

Array Descriptor for Narrow Band Matrices

DESC_()	Symbolic Name	Scope	Definition
1	DTYPE_A	(global)	Descriptor type DTYPE_A=501 for 1 x Pc process grid for band and tridi agonal matrices block-column distributed.
2	CTXT_A	(global)	BLACS context handle.
3	N_A	(global)	Number of columns in global array A.
4	NB_A	(global)	Blocking factor used to distribute the columns of array A.
5	CSRC_A	(global)	Process column over which the first column of the array A is distributed.
6	LLD_A	(local)	Leading dimension of the local array. For the tridiagonal subroutines, this entry is ignored.
7	–	–	Unused, reserved.

Array Descriptor for Right Hand Sides for Narrow Band Linear Solvers

DESC_()	Symbolic Name	Scope	Definition
1	DTYPE_B	(global)	Descriptor type DTYPE_B=502 for Pr x 1 process grid for block-row distributed matrices .
2	CTXT_B	(global)	BLACS context handle.
3	M_B	(global)	Number of rows in global array B
4	MB_B	(global)	Blocking factor used to distribute the rows of array B.
5	RSRC_B	(global)	Process row over which the first row of the array B is distributed.
6	LLD_B	(local)	Leading dimension of the local array. For the tridiagonal subroutines, this entry is i gnored.
7	–	–	Unused, reserved.

On line tutorial: *http://acts.nersc.gov/scalapack/hands-on/main.html*

Hands-On Exercises for ScaLAPACK

ScaLAPACK Team
August 2002

Introduction

These exercises provide basic and more advanced programming instruction for writing parallel programs calling the BLACS, PBLAS, and ScaLAPACK. A basic knowledge of Fortran, parallel programming with message-passing, and MPI are assumed. Some of the exercises also require an understanding of two-dimensional block cyclic data distribution.

Detailed information on the BLACS, PBLAS, and ScaLAPACK may be found at the respective URLs:

- http://www.netlib.org/blacs
- http://www.netlib.org/scalapack
- http://www.netlib.org/scalapack/pblas_qref.html

Exercises 1 and 2 give an introduction to parallel programming with the Basic Linear Algebra Communication Subprograms (BLACS). Exercises 3, 4, and 5 provide a range of simplistic to more complex parallel programs calling ScaLAPACK and PBLAS. More example programs for ScaLAPACK can be found at http://www.netlib.org/scalapack/examples.

The instructions for the exercises assume that the underlying system is an IBM SP or a Cray T3E; using up to six processes that do message-passing. These example programs use MPI as the underlying message-passing layer. The version of MPI used in these examples is the version 3.0, and we assume the user has this version installed.

These hands-on exercises were prepared in collaboration with the Joint Institute for Computational Science at the University of Tennessee, based on contributions from A. YarKhan, C. Hastings, S. Blackford, C. Whaley, A. Petitet and O. Marques.

Exercise 1: **BLACS - Hello World Example**
Exercise 2: **BLACS - Pi Example**
Exercise 3: **ScaLAPACK - Example Program 1**
Exercise 4: **ScaLAPACK - Example Program 2**
Exercise 5: **PBLAS Example**

Addition Resources:

- Block Cyclic Data Distribution
- Useful calling sequences
- Download all exercises

Exercise 3: ScaLAPACK - Example Program 1

Information: This is a very simplistic example program that solves a linear system by calling the ScaLAPACK routine PSGESV. More complete details of this example program can be found in Chapter 2 of the ScaLAPACK Users' Guide. This example program demonstrates the basic requirements to call a ScaLAPACK routine: initializing the process grid, assigning the matrix to the processes, calling the ScaLAPACK routine, and releasing the process grid.

This example program solves the 9-by-9 system of linear equations given by:

$$\begin{pmatrix} 19 & 3 & 1 & 12 & 1 & 16 & 1 & 3 & 11 \\ -19 & 3 & 1 & 12 & 1 & 16 & 1 & 3 & 11 \\ -19 & -3 & 1 & 12 & 1 & 16 & 1 & 3 & 11 \\ -19 & -3 & -1 & 12 & 1 & 16 & 1 & 3 & 11 \\ -19 & -3 & -1 & -12 & 1 & 16 & 1 & 3 & 11 \\ -19 & -3 & -1 & -12 & -1 & 16 & 1 & 3 & 11 \\ -19 & -3 & -1 & -12 & -1 & -16 & 1 & 3 & 11 \\ -19 & 3 & -1 & -12 & -1 & -16 & -1 & 3 & 11 \\ -19 & -3 & -1 & -12 & -1 & -16 & -1 & -3 & 11 \end{pmatrix} \begin{pmatrix} x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \\ x8 \\ x9 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

using the ScaLAPACK driver routine PSGESV. The ScaLAPACK routine PSGESV solves a system of linear equations $A*X = B$, where the coefficient matrix (denoted by A) and the right-hand-side matrix (denoted by B) are real, general distributed matrices. The coefficient matrix A is distributed as depicted below and, for simplicity, we shall solve the system for one right-hand side (NRHS=1); that is, the matrix B is a vector. The third element of the matrix B is equal to 1, and all other elements are equal to 0. After solving this system of equations, the solution vector X is given by

$$\begin{pmatrix} x1 \\ x2 \\ x3 \\ x4 \\ x5 \\ x6 \\ x7 \\ x8 \\ x9 \end{pmatrix} = \begin{pmatrix} 0 \\ -1/6 \\ 1/2 \\ 0 \\ 0 \\ 0 \\ -1/2 \\ 1/6 \\ 0 \end{pmatrix}$$

Let us assume that the matrix A is partitioned and distributed such that we have chosen the row and column block sizes as MB=NB=2, and the matrix is distributed on a 2-by-3 process grid (P_r=2, P_c=3). The partitioning and distribution of our example matrix A is represented in the two figures below, where, to aid visualization, we use the notation s=19, c=3, a=1, l=12, p=16, and k=11.

Partioning of global matrix A:

s	c	a	l	a	p	a	c	k
-s	c	a	l	a	p	a	c	k
-s	-c	a	l	a	p	a	c	k
-s	-c	-a	l	a	p	a	c	k
-s	-c	-a	-l	a	p	a	c	k
-s	-c	-a	-l	-a	p	a	c	k
-s	-c	-a	-l	-a	-p	a	c	k
-s	c	-a	-l	-a	-p	-a	c	k
-s	-c	-a	-l	-a	-p	-a	-c	k

Mapping of matrix A onto process grid (P_r=2, P_c=3). Note, for example, that process (0,0) contains a local array of size $A(5,4)$.

0			1			2		
s	c	a	c	a	l	k	a	p
-s	c	a	c	a	l	k	a	p
-s	-c	a	c	-a	-l	k	a	p
-s	-c	a	c	-a	-l	k	-a	p
-s	-c	-a	-c	-a	-l	k	-a	-p
-s	-c	a	c	a	l	k	a	p
-s	-c	a	c	-a	l	k	a	p
-s	-c	a	c	-a	-l	k	-a	-p
-s	c	-a	c	-a	-l	k	-a	-p

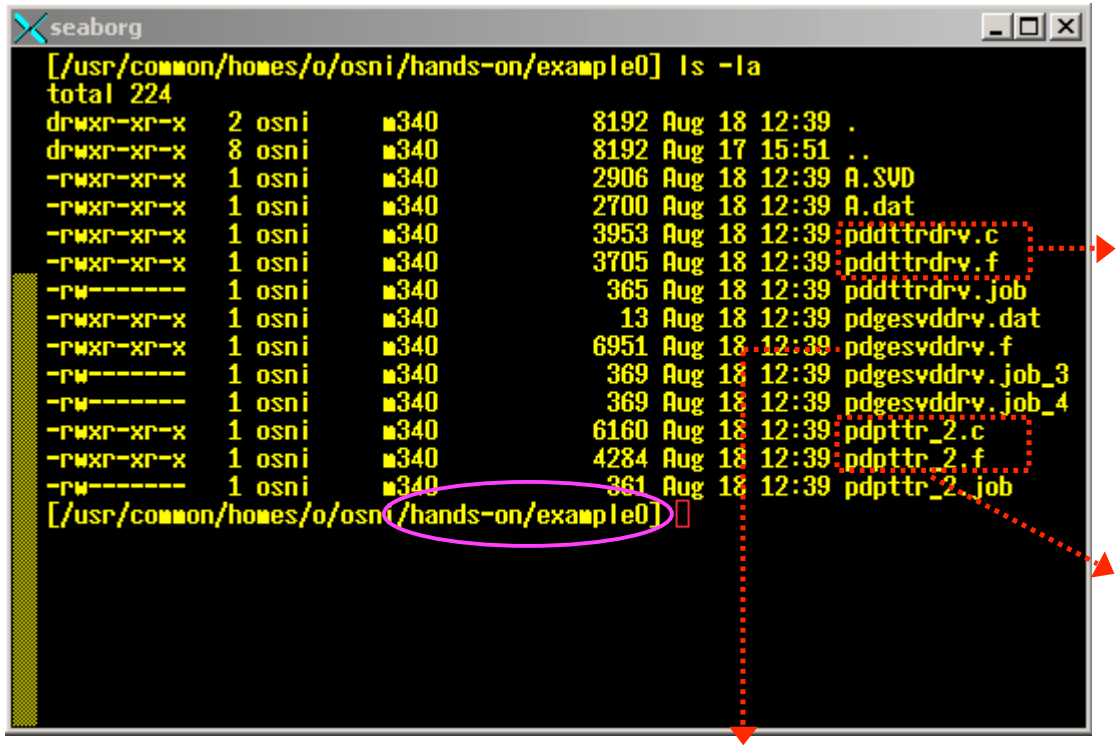
0

1

```
      PROGRAM PSGESVDRIVER
*
*     Example Program solving Ax=b via ScaLAPACK routine PSGESV
*
*     .. Parameters ..

*     ..
*     .. Local Scalars ..

*     ..
*     .. Local Arrays ..

*     .. Executable Statements ..
*
*     INITIALIZE THE PROCESS GRID
*
      CALL SL_INIT( ICTXT, NPROW, NPCOL )
      CALL BLACS_GRIDINFO( ICTXT, NPROW, NPCOL, MYROW, MYCOL )
*
*     If I'm not in the process grid, go to the end of the program
*
      IF( MYROW.EQ.-1 )
     $   GO TO 10
*
*     DISTRIBUTE THE MATRIX ON THE PROCESS GRID
*     Initialize the array descriptors for the matrices A and B
*
      CALL DESCINIT( DESCA, M, N, MB, NB, RSRC, CSRC, ICTXT, MXLLDA,
     $               INFO )
      CALL DESCINIT( DESCB, N, NRHS, NB, NBRHS, RSRC, CSRC, ICTXT,
     $               MXLLDB, INFO )
*
*     Generate matrices A and B and distribute to the process grid
*
      CALL MATINIT( A, DESCA, B, DESCB )
*
*     Make a copy of A and B for checking purposes
*
      CALL PSLACPY( 'All', N, N, A, 1, 1, DESCA, A0, 1, 1, DESCA )
      CALL PSLACPY( 'All', N, NRHS, B, 1, 1, DESCB, B0, 1, 1, DESCB )
*
*     CALL THE SCALAPACK ROUTINE
*     Solve the linear system A * X = B
*
      CALL PSGESV( N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB,
     $             INFO )
```

M

Contents of hands-on/example0



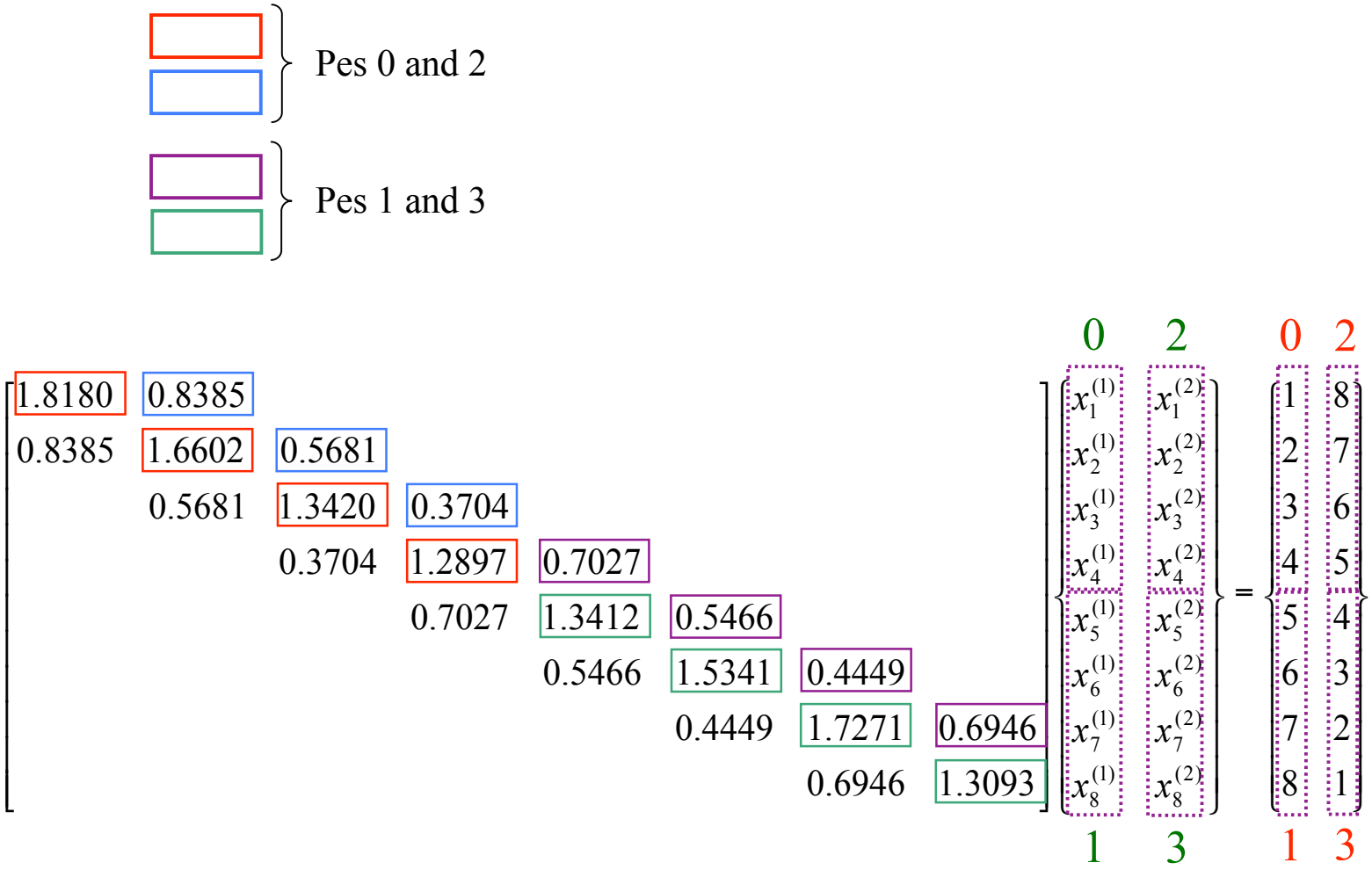
pddttrdrv.c (*pddttrdrv.f*): illustrates the use of the ScaLAPACK routines PDDTTRF and PDDTTRS to factor and solve a (diagonally dominant) tridiagonal system of linear equations $Tx = b$. After compilation, it can be executed with *llsubmit pddttrdrv.job*.

pdpttr_2.c (*pdpttr_2.f*): illustrates the use of the ScaLAPACK routines PDPTTRF and PPPTTRS to factor and solve a symmetric positive definite tridiagonal system of linear equations $Tx = b$, in two distinct contexts. After compilation, it can be executed with *llsubmit pdpttr_2.job*.

pdgesvddrv.f: reads a (full) matrix A from a file, distributes A among the available processors and then call the ScaLAPACK subroutine PDGESVD to computed the SVD of A, $A=USV^T$. It requires the file *pdgesvddrv.dat*, which should contain: line 1, the name of the file where A will be read from; line 2, the number of rows of A; line 3: the number of columns of A. Considering the file *A.dat*:

- if $m=n=10$ the results are given in the file *A.SVD*
- if $m=10$, $n=7$: diag(S)=[*4.4926 1.4499 0.8547 0.8454 0.6938 0.4332 0.2304*]
- if $m=7$, $n=10$: diag(S)=[*4.5096 1.1333 1.0569 0.8394 0.8108 0.5405 0.2470*]

Data distribution for *pdpttr_2.c* (*pdpttr_2.f*)



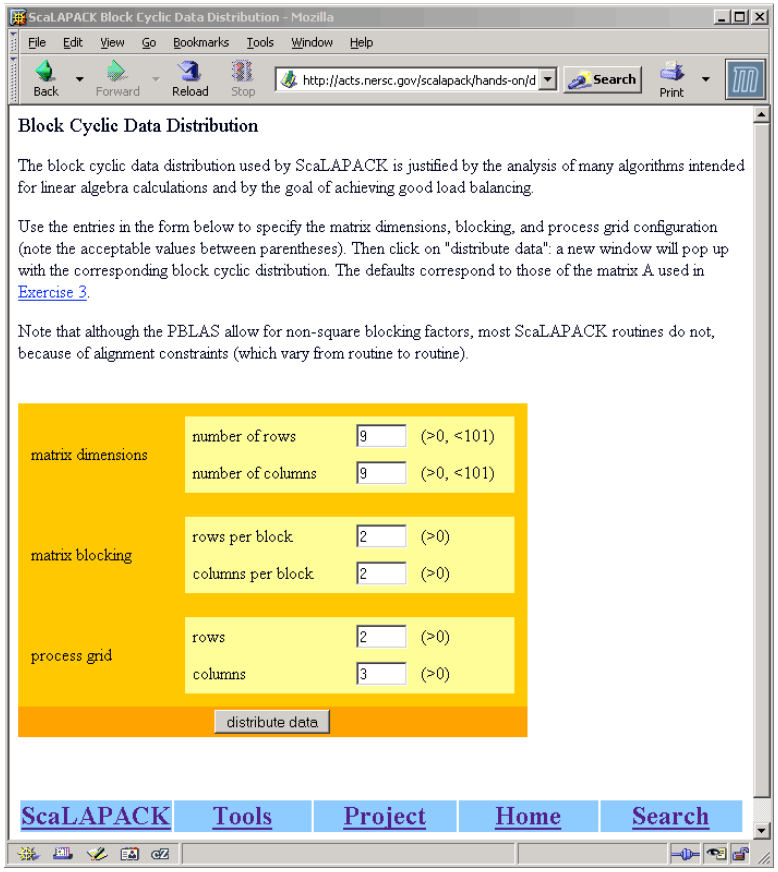
```
/**********************************************************************
/
/* This program illustrates the use of the ScaLAPACK routines PDPTTRF
*/
/* and PPPTTRS to factor and solve a symmetric positive definite
*/
/* tridiagonal system of linear equations, i.e., T*x = b, with
*/
/* different data in two distinct contexts.
*/
/**********************************************************************
/

/* a bunch of things omitted for the sake of space */

main()
{

   /* Start BLACS */
   Cblacs_pinfo( &mype, &npe );
   Cblacs_get( 0, 0, &context );
   Cblacs_gridinit( &context, "R", 1, npe );
   /* Processes 0 and 2 contain d(1:4) and e(1:4) */
   /* Processes 1 and 3 contain d(5:8) and e(5:8) */
   if      ( mype == 0 || mype == 2 ){
           d[0]=1.8180; d[1]=1.6602; d[2]=1.3420; d[3]=1.2897;
           e[0]=0.8385; e[1]=0.5681; e[2]=0.3704; e[3]=0.7027;
   }
   else if ( mype == 1 || mype == 3 ){
           d[0]=1.3412; d[1]=1.5341; d[2]=1.7271; d[3]=1.3093;
           e[0]=0.5466; e[1]=0.4449; e[2]=0.6946; e[3]=0.0000;
   }
   if ( mype == 0 || mype == 1 ) {
      /* New context for processes 0 and 1 */
      map[0]=0; map[1]=1;
      Cblacs_get( context, 10, &context_1 );
      Cblacs_gridmap( &context_1, map, 1, 1, 2 );
      /* Right-hand side is set to b = [ 1 2 3 4 5 6 7 8 ] */
      if      ( mype == 0 ) {
              b[0]=1.0; b[1]=2.0; b[2]=3.0; b[3]=4.0;
      }
      else if ( mype == 1 ) {
              b[0]=5.0; b[1]=6.0; b[2]=7.0; b[3]=8.0;
      }
      /* Array descriptor for A (D and E) */
      desca[0]=501; desca[1]=context_1; desca[2]=n; desca[3]=nb;
      desca[4]=0; desca[5]=lda; desca[6]=0;
      /* Array descriptor for B */
      descb[0]=502; descb[1]=context_1; descb[2]=n; descb[3]=nb;
      descb[4]=0; descb[5]=ldb; descb[6]=0;
      /* Factorization */
      pdpttrf( &n, d, e, &ja, desca, af, &laf,
               work, &lwork, &info );
      /* Solution */
      pdpttrs( &n, &nrhs, d, e, &ja, desca, b, &ib, descb,
```

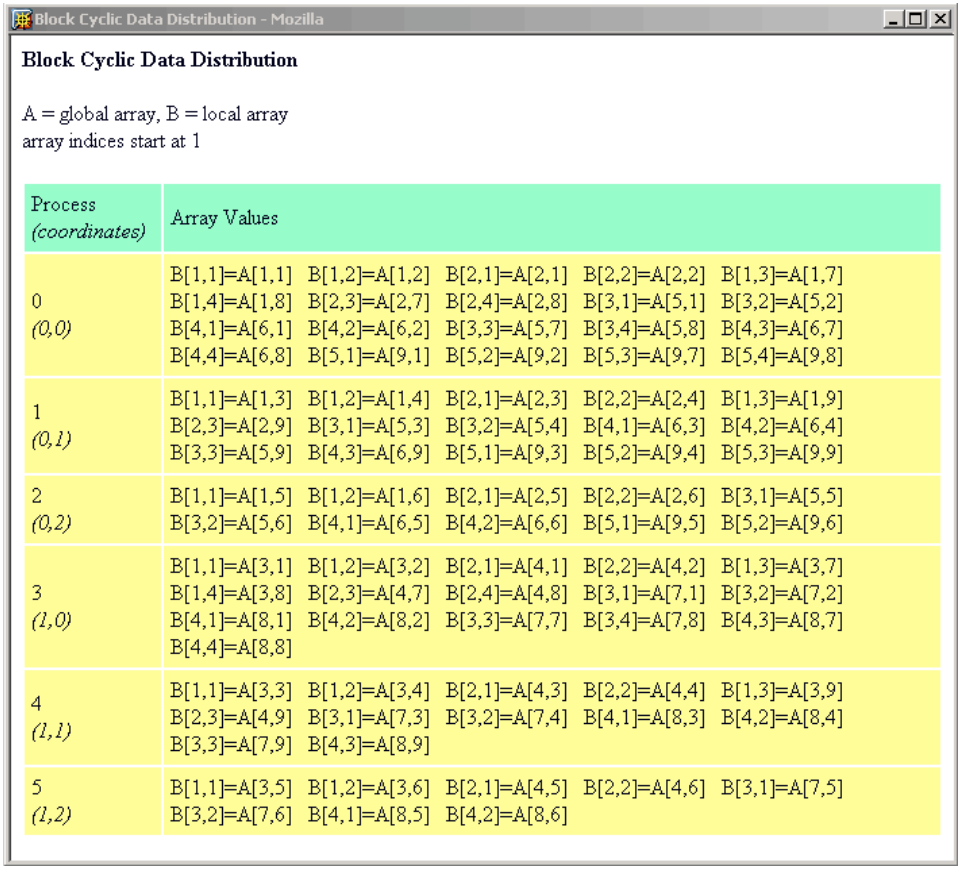
```
else {
      /* New context for processes 0 and 1 */
      map[0]=2; map[1]=3;
      Cblacs_get( context, 10, &context_2 );
      Cblacs_gridmap( &context_2, map, 1, 1, 2 );
      /* Right-hand side is set to b = [ 8 7 6 5 4 3 2 1 ] */
      if      ( mype == 2 ) {
              b[0]=8.0; b[1]=7.0; b[2]=6.0; b[3]=5.0;
      }
      else if ( mype == 3 ) {
              b[0]=4.0; b[1]=3.0; b[2]=2.0; b[3]=1.0;
      }
      /* Array descriptor for A (D and E) */
      desca[0]=501; desca[1]=context_2; desca[2]=n; desca[3]=nb;
      desca[4]=0; desca[5]=lda; desca[6]=0;
      /* Array descriptor for B */
      descb[0]=502; descb[1]=context_2; descb[2]=n; descb[3]=nb;
      descb[4]=0; descb[5]=ldb; descb[6]=0;
      /* Factorization */
      pdpttrf( &n, d, e, &ja, desca, af, &laf,
               work, &lwork, &info );
      /* Solution */
      pdpttrs( &n, &nrhs, d, e, &ja, desca, b, &ib, descb,
               af, &laf, work, &lwork, &info );
      printf( "MYPE=%i: x[:] = %7.4f %7.4f %7.4f %7.4f\n",
              mype, b[0], b[1], b[2], b[3]);
   }
   Cblacs_gridexit( context );
   Cblacs_exit( 0 );
}
```

```
Using Matlab notation:

T = diag(D)+diag(E,-1)+diag(E,1)

where

D = [ 1.8180 1.6602 1.3420 1.2897 1.3412 1.5341 1.7271 1.3093
]
E = [ 0.8385 0.5681 0.3704 0.7027 0.5466 0.4449 0.6946 ]

Then, solving T*x = b,

if b = [ 1 2 3 4 5 6 7 8 ]
x = [ 0.3002 0.5417 1.4942 1.8546 1.5008 3.0806 1.0197 5.5692
]

if b = [ 8 7 6 5 4 3 2 1 ]
x = [ 3.9036 1.0772 3.4122 2.1837 1.3090 1.2988 0.6563 0.4156
]
```

2D Block-Cyclic Distribution

http://acts.nersc.gov/scalapack/hands-on/datadist.html





Process *(coordinates)*	Array Values
0 *(0,0)*	B[1,1]=A[1,1] B[1,2]=A[1,2] B[2,1]=A[2,1] B[2,2]=A[2,2] B[1,3]=A[1,7] B[1,4]=A[1,8] B[2,3]=A[2,7] B[2,4]=A[2,8] B[3,1]=A[5,1] B[3,2]=A[5,2] B[4,1]=A[6,1] B[4,2]=A[6,2] B[3,3]=A[5,7] B[3,4]=A[5,8] B[4,3]=A[6,7] B[4,4]=A[6,8] B[5,1]=A[9,1] B[5,2]=A[9,2] B[5,3]=A[9,7] B[5,4]=A[9,8]
1 *(0,1)*	B[1,1]=A[1,3] B[1,2]=A[1,4] B[2,1]=A[2,3] B[2,2]=A[2,4] B[1,3]=A[1,9] B[2,3]=A[2,9] B[3,1]=A[5,3] B[3,2]=A[5,4] B[4,1]=A[6,3] B[4,2]=A[6,4] B[3,3]=A[5,9] B[4,3]=A[6,9] B[5,1]=A[9,3] B[5,2]=A[9,4] B[5,3]=A[9,9]
2 *(0,2)*	B[1,1]=A[1,5] B[1,2]=A[1,6] B[2,1]=A[2,5] B[2,2]=A[2,6] B[3,1]=A[5,5] B[3,2]=A[5,6] B[4,1]=A[6,5] B[4,2]=A[6,6] B[5,1]=A[9,5] B[5,2]=A[9,6]
3 *(1,0)*	B[1,1]=A[3,1] B[1,2]=A[3,2] B[2,1]=A[4,1] B[2,2]=A[4,2] B[1,3]=A[3,7] B[1,4]=A[3,8] B[2,3]=A[4,7] B[2,4]=A[4,8] B[3,1]=A[7,1] B[3,2]=A[7,2] B[4,1]=A[8,1] B[4,2]=A[8,2] B[3,3]=A[7,7] B[3,4]=A[7,8] B[4,3]=A[8,7] B[4,4]=A[8,8]
4 *(1,1)*	B[1,1]=A[3,3] B[1,2]=A[3,4] B[2,1]=A[4,3] B[2,2]=A[4,4] B[1,3]=A[3,9] B[2,3]=A[4,9] B[3,1]=A[7,3] B[3,2]=A[7,4] B[4,1]=A[8,3] B[4,2]=A[8,4] B[3,3]=A[7,9] B[4,3]=A[8,9]
5 *(1,2)*	B[1,1]=A[3,5] B[1,2]=A[3,6] B[2,1]=A[4,5] B[2,2]=A[4,6] B[3,1]=A[7,5] B[3,2]=A[7,6] B[4,1]=A[8,5] B[4,2]=A[8,6]

Global Arrays (GA) wrappers

http://www.emsl.pnl.gov/docs/global/ga.html

- Simpler than message-passing for many applications
- Complete environment for parallel code development
- Data locality control similar to distributed memory/message passing model
- Compatible with MPI
- Scalable

Distributed Data: data is explicitly associated with each processor, accessing data requires specifying the location of the data on the processor and the processor itself.

Shared Memory: data is an a globally accessible address space, any processor can access data by specifying its location using a global index.

GA: distributed dense arrays that can be accessed through a shared memory-like style.

The function

```
Fortran   integer function ga_solve(g_a, g_b)
C         int GA_Solve(int g_a, int g_b)
C++       int GA::GlobalArray::solve(const GA::GlobalArray * g_a)
```

solves a system of linear equations $A * X = B$. It first will call the Cholesky factorization routine and, if successful, will solve the system with the Cholesky solver. If Cholesky is not able to factorize A, then it will call the LU factorization routine and will solve the system with forward/backward substitution. On exit B will contain the solution X.

Conclusions

- Different mechanisms for supporting the distribution of data are needed
- Applications require different levels of abstraction

MS18:
- pMatlab
- Star-P
- –

PyACTS:
- Choice of scripting language: Python
- Uses PyMPI and Numeric
- Intended for testing and not high-performing production runs.